Implementation of Class Based Queueing in Linux 2.0.30

Loh Kok Jeng
{elelohkj@leonis.nus.edu.sg}
Singapore Linux Conference '99
1-5 March 1999

Tutorial Outline

- Overview of CBQ
- Linux implementation of CBQ
- Performance of CBQ
  - Overhead of CBQ
  - Bandwidth guarantee
  - Delay, delay jitter and packet loss
- Conclusion
- References

Introduction on CBQ

- It is a **link sharing mechanism** which tries to provide **real-time service**.
- It consists of **classes**, each associated with certain **share of bandwidth** and a **priority**.
- It consists of a **root class**, **interior classes** and **leaf classes** defined in a hierarchy. Only **leaf classes** have physical queues.
- A **flow** or group of flows are assigned to a class.
- It provides **isolation** among classes of traffic while allows them to share the bandwidth of the link.
- Each class should receive roughly its allocated bandwidth over some interval of time, even during congestion.

Introduction on CBQ

- Link-sharing provides a means for sharing the bandwidth of a link among different organizations, protocols, etc.
- When some classes are not using their allocated bandwidth, the distribution of the excess bandwidth among the other classes follows some appropriate set of guidelines.
- Classes with higher priority are served first. Therefore, the packets that belong to these classes generally experience lower queueing delay.
- Priority-based scheduling can be used to reduce delay for the real-time traffic.
- Priority-based scheduling alone causes starvation. Link-sharing mechanism needed to serve the classes in accordance to their allocated bandwidth and prevent certain classes from hijacking the bandwidth.

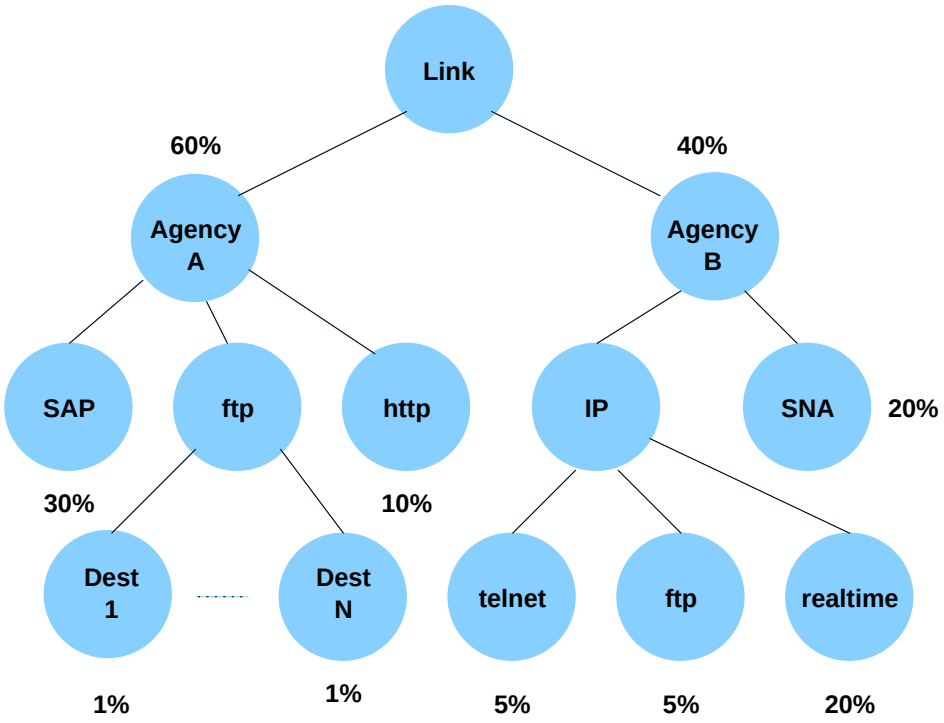
Introduction on CBQ

- General scheduler schedules packets from leaf classes without regard to link-sharing guidelines.
- General scheduler:-
  - Packet round-robin
  - Weighted round-robin
- Link-sharing schedules packets from some leaf classes that have been exceeding their link-sharing allocation in times of congestion.

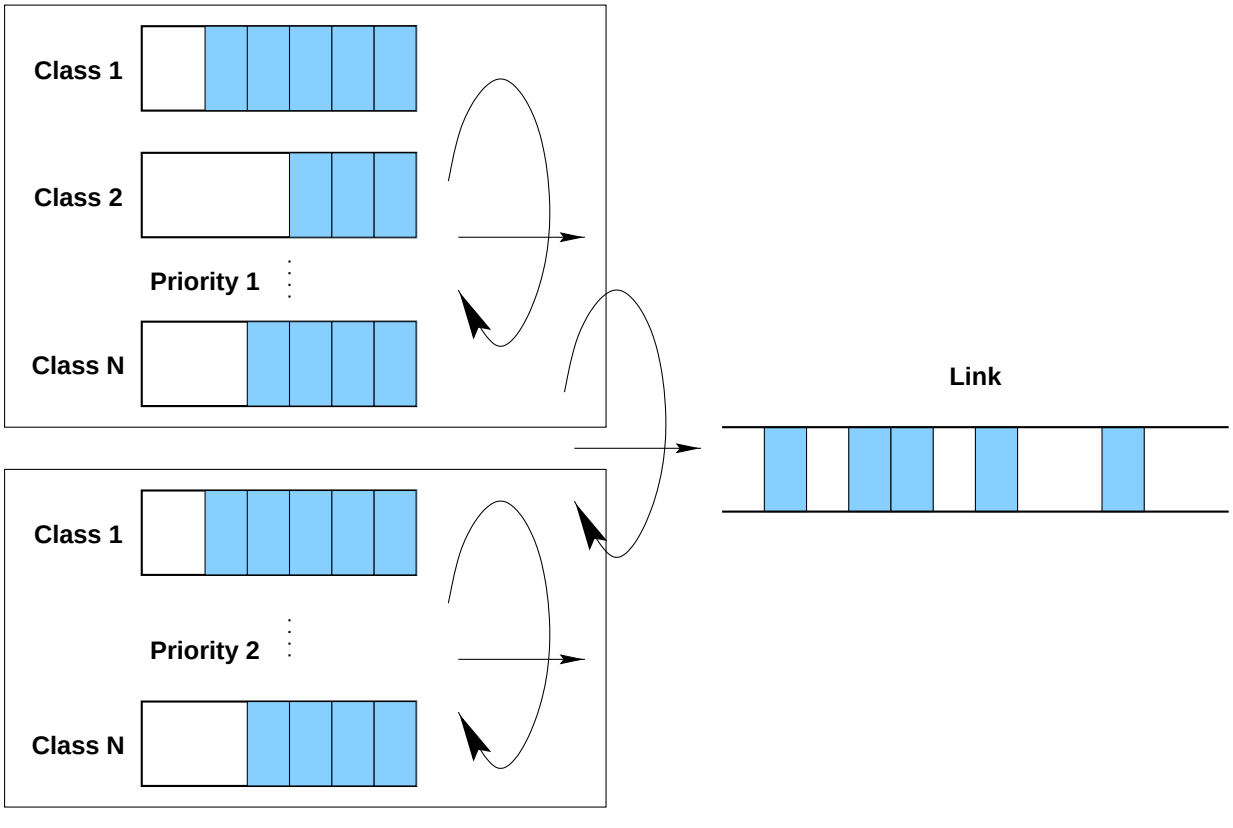
Introduction on CBQ

- Link-sharing guidelines:-
  - **Formal link-sharing guideline** (i) The class is not overlimit OR (ii) The class has a not-overlimit ancestor at level i, and there are no unsatisfied class in the link-sharing structure at levels lower than i.
  - **Ancestor-only guideline** (i) The class is not overlimit OR (ii) the class has an underlimit ancestor.
  - **Top-level guideline** (i) The class is not overlimit OR (ii) the class has an underlimit ancestor whose level is at most Top-Level.

Link-Sharing Mechanism of CBQ



Priority Based Scheduling of CBQ

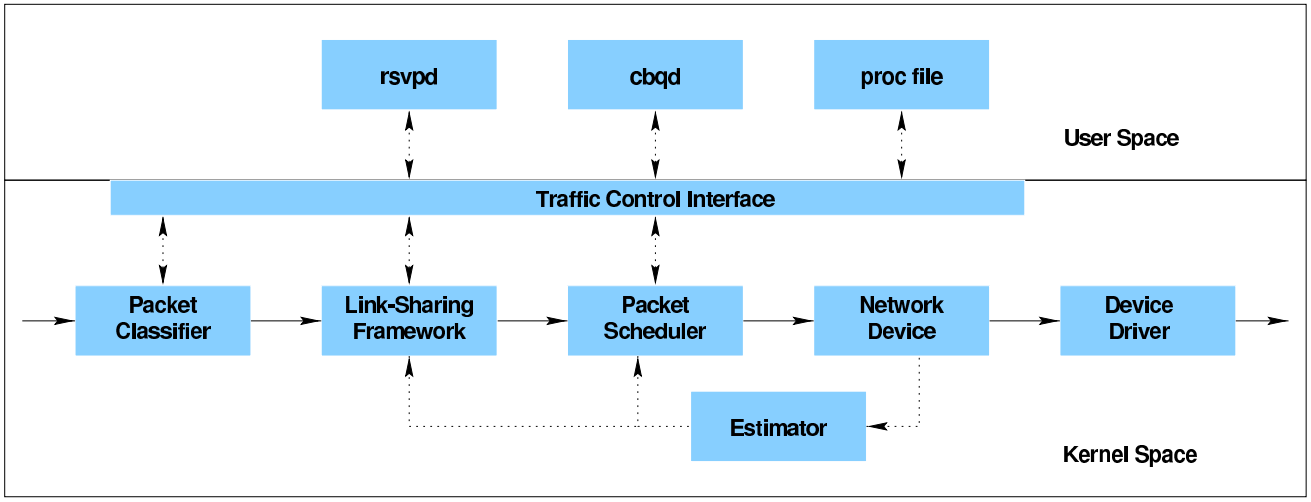


Components of CBQ

- Implemented at outgoing interface.
- **Packet classifier** maps packets arriving at the gateway to the appropriate classes for that output link.
- **Link-sharing framework** maintains link-sharing constraints.
- **Packet Scheduler** schedules classes according to their bandwidth allocation and priority.
- **Estimator** estimates the bandwidth used by each class over an appropriate time interval.
- **Traffic control interface** provides a means for setting, and configuring classes, creating filters at classifier, and collection of statistics.

Components of CBQ

- **Network device** is an abstraction layer that hides the complexity and differences of device drivers from the kernel.



Linux Implementation of CBQ

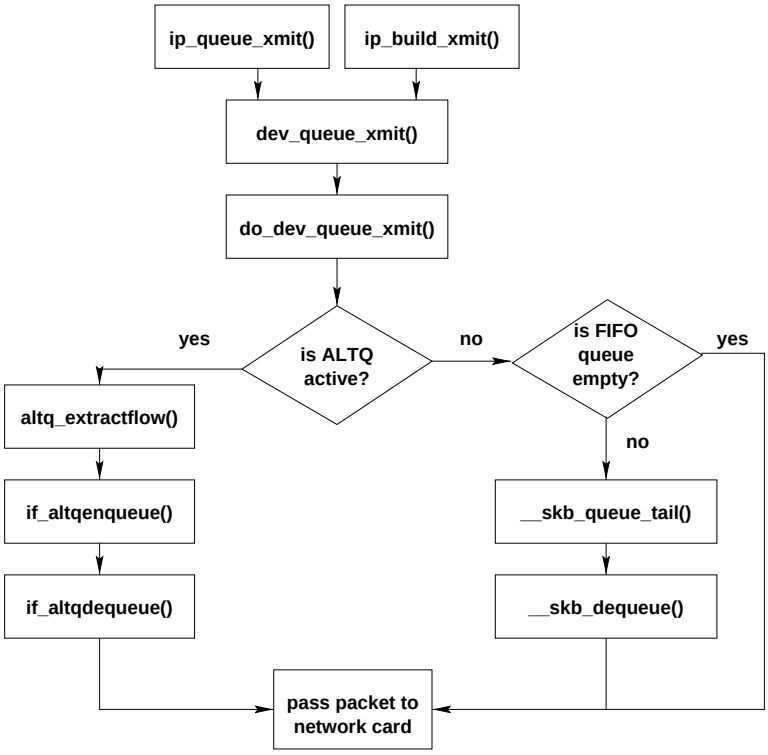
- A port of **ALTQ** from FreeBSD.
- ALTQ provides a framework that allows other packet schedulers to be implemented in the kernel.
- A wrapper for ALTQ is implemented in **altq.c**. It provides functions for activating and disabling certain packet scheduler; extracting flow information; and interface to device driver of packet schedulers.
- ALTQ and other packet schedulers are controlled through a character device (e.g. /dev/altq and /dev/cbq). **Open**, **close** and **ioctl** are used by user space program to control ALTQ and packet schedulers.
- **dev.c** and **device** data structure are modified to divert packets to ALTQ module.

Device Data Structure

```
struct device {
    ......
    ......
    struct if_statistic* (*get_wireless_stats)(struct device *dev);

#ifdef CONFIG_ALTQ
    int     if_altqflags;      /* altq flags                 */
    void    *if_altqp;         /* pointer to altq state data */
    int     (*if_altqenqueue)(struct device *, struct sk_buff *,
                              struct flowinfo *, int);
    struct sk_buff* (*if_altqdequeue(struct device *, int);
#endif
}
```

Flow Chart



Linux Implementation of CBQ

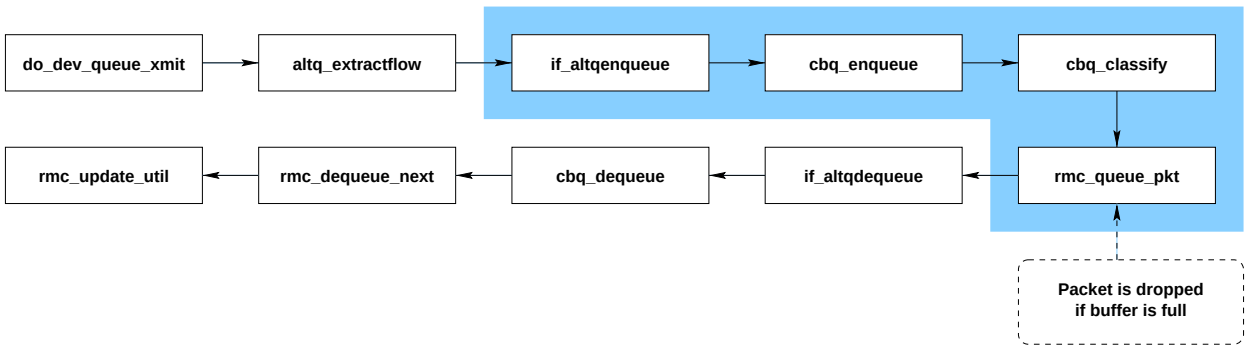
- Interface to CBQ is implemented in **cbq.c**. Provides functions for creating and removing classes as well as filters; getting statistics; enqueueing and dequeueing packets.
- Packet classifier is implemented in **cbq_class.c**. Flows are identified by their source address and port, together with destination address and port.
- **rm_class.c** performs the actual packet scheduling.
- **rmc_dequeue_next** performs WRR on classes of the same priority. Each class is assigned a weight that is proportional to the its bandwidth allocation. The weight determines the number of bytes that a class is allowed to send each round. If a class sent more than its share, it will be penalized in future rounds.

Linux Implementation CBQ

- When a class is overlimit, it is prohibited from sending any packet. **rmc_delay_action** is invoked to calculate the next time that the class is allowed to transmit again.
- Exponential Weighted Moving Average is used to estimate the bandwidth consumption of each class. This estimating function is implemented in **rmc_update_util**.

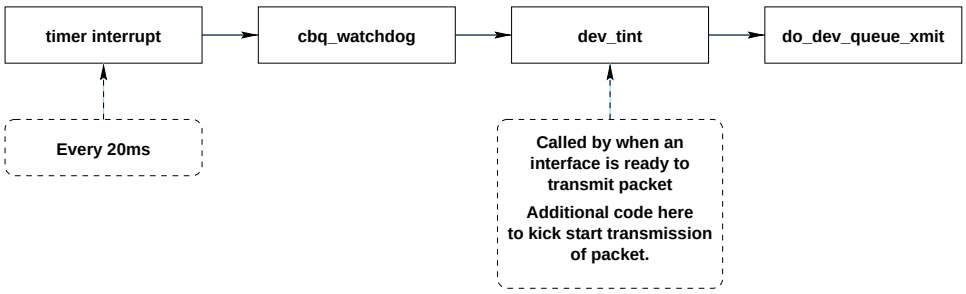
Transmission of Packet

- The information of the packet is extracted by **altq_extractflow**.
- Packet classified by **cbq_classify**.
- Packet inserted into the appropriate queue by **rmc_queue_pkt**.
- Check if there is any packet ready for transmission (**rmc_dequeue_next**).
- Update the bandwidth usage (**rmc_update_util**).



Linux Implementation of CBQ

- Timer interrupt is invoked every $20ms$.
- Call **cbq_watchdog** which in turn calls **dev_tint**.
- **dev_tint** modified to check if there is packet to transmit.



Sample Content of Proc File

- A **proc** file is implemented for CBQ.
- Allows easy access to the status of CBQ from user space.
- Contains statistics of each class.

			sent		dropped			
dev	cls	prty	pkts	bytes	pkts	bytes	ns_byte	qcnt
eth0	rt	0	0	0	0	0	833	0
eth0	dft	3	2	129	0	0	1176	0
eth0	ctl	6	0	0	0	0	40000	0
eth0	0	6	0	0	0	0	2666	0

Sample Config File

- Create 2 classes:– res_class and unres_class.
- res_class:- 50% of bandwidth, priority 6, used by RSVP to make reservation.
- unres_class:- 50% of bandwidth, priority 3, used by best-effort traffic.

```
interface eth0 bandwidth 10000000 cbq
class cbq eth0 root_class null priority 0 admission none pbandwidth 100
class cbq eth0 res_class root_class priority 6 pbandwidth 50 admission cntlload
        borrow root_class
class cbq eth0 unres_class root_class priority 3 pbandwidth 50 default
        borrow root_class
```


cbqd – Simple CBQ daemon

- Can be run in background or interactive mode.

```
Enter ? or command:
cbq > ?
DoCommand: ?
Commands are:
 help | ?
 quit
 interface if_name [bandwidth bps] [sched_type]
 class cbq if_name class_name parent [borrow borrow_class]
        [admission ctlload|none] [maxburst count] [minburst count]
        [packetsize bytes] [priority pri] [pbandwidth percent]
 filter if_name class_name dst [netmask #] dport src [netmask #] sport proto
 cbq if_name {enable|disable|acc enable|acc disable}
cbq >
```

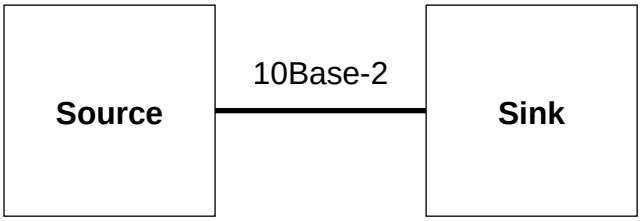
Implementation Issues

- Allow choice between FIFO and CBQ.
- New queueing discipline can be added to the ALTQ framework by adding a new entry in the ALTQ device table.
- Interrupts must be blocked during dequeue process to prevent race condition.
- The completion time of packet is estimated rather than using a callback method.
- A timer is used to periodically check whether regulated class has been allowed to transmit again.

Implementation Issues

- Timer granularity. Packet completion time calculated based on **do_gettimeofday**.
- An overlimit class is suspended by CBQ. Resumption of transmission is triggered by either a send/receive event, or a timer interrupt which has granularity of 20ms. In the worst case, the timing will be rounded up by at least 20ms.

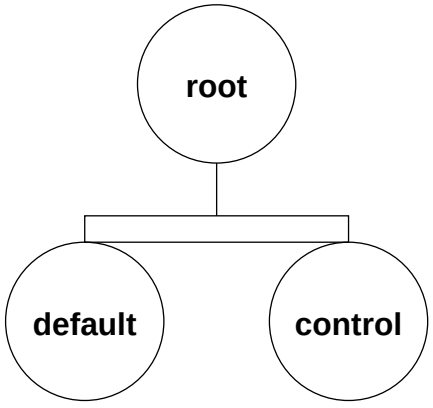
Network Setup for Testing CBQ at End-hosts



Pentium 90MHz
24 Mbytes RAM
Etherlink III Combo ISA
(3C509B)

Pentium 90MHz
24 Mbytes RAM
Etherlink III Combo ISA
(3C509B)

Classes for Testing Overhead of CBQ



Effect of CBQ on Throughput

- TCP

Request size (bytes)	Throughput (Mbps) (CBQ)	Throughput (Mbps) (no CBQ)
128	8.41	8.40
256	8.38	8.40
512	8.43	8.40
1024	8.41	8.41

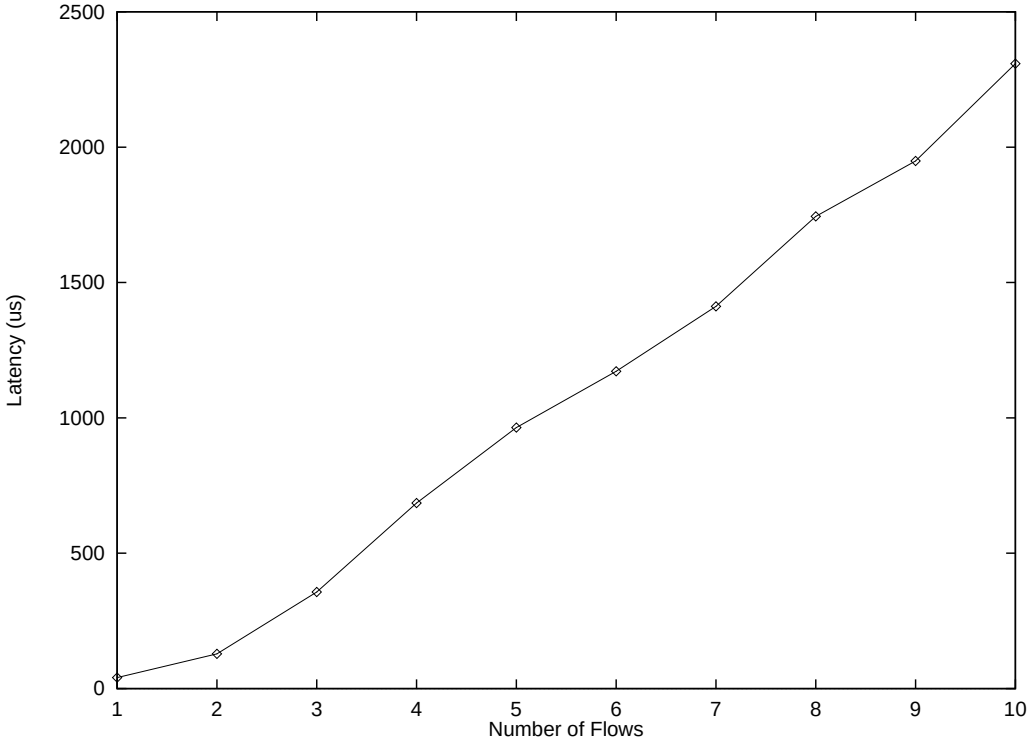
- UDP

Packet size (bytes)	Throughput (Mbps) (CBQ)	Throughput (Mbps) (no CBQ)
128	6.57	6.56
256	7.93	7.93
512	8.84	8.84
1024	9.39	9.39

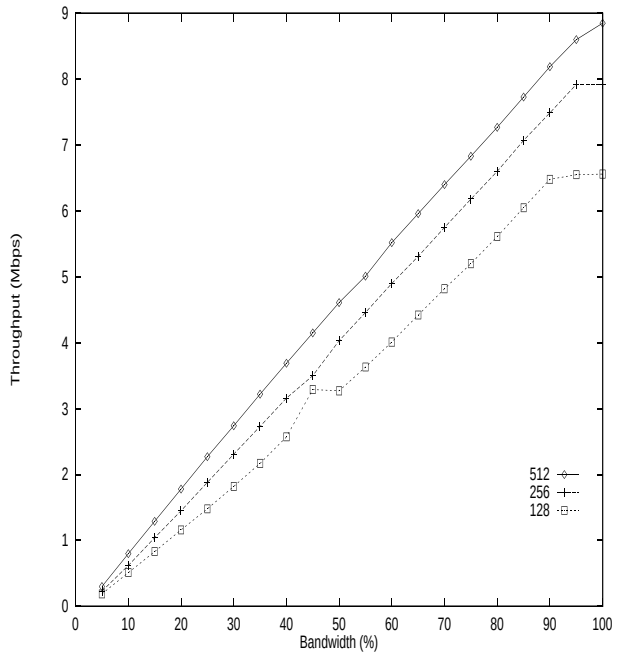
Latency Caused by CBQ

Request size (bytes)	Response size (bytes)	with CBQ RTT (μsec)	no CBQ RTT (μsec)	Latency (μsec)
1	1	625.9	597.4	28.5
64	64	806.1	785.4	20.7
128	64	912.9	886.0	26.9
256	64	1132.7	1094.3	38.4
512	64	1533.9	1495.3	38.5
1025	64	2373.0	2348.1	24.9

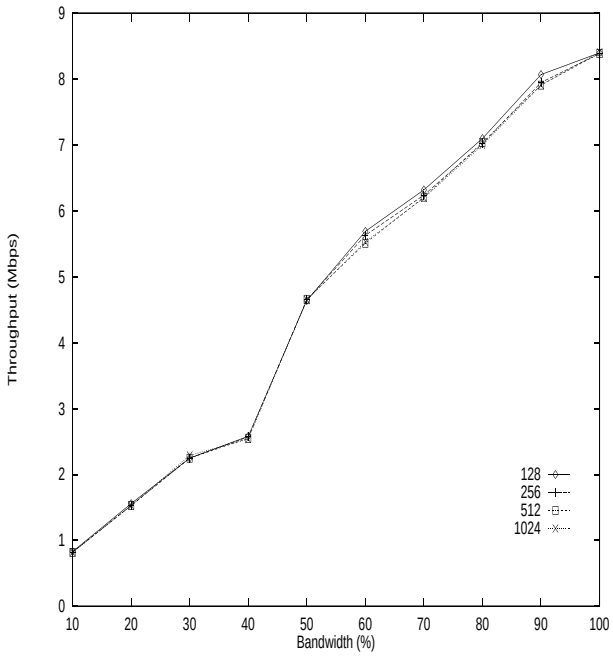
Latency Incurred by CBQ



Throughput vs. Bandwidth Allocation

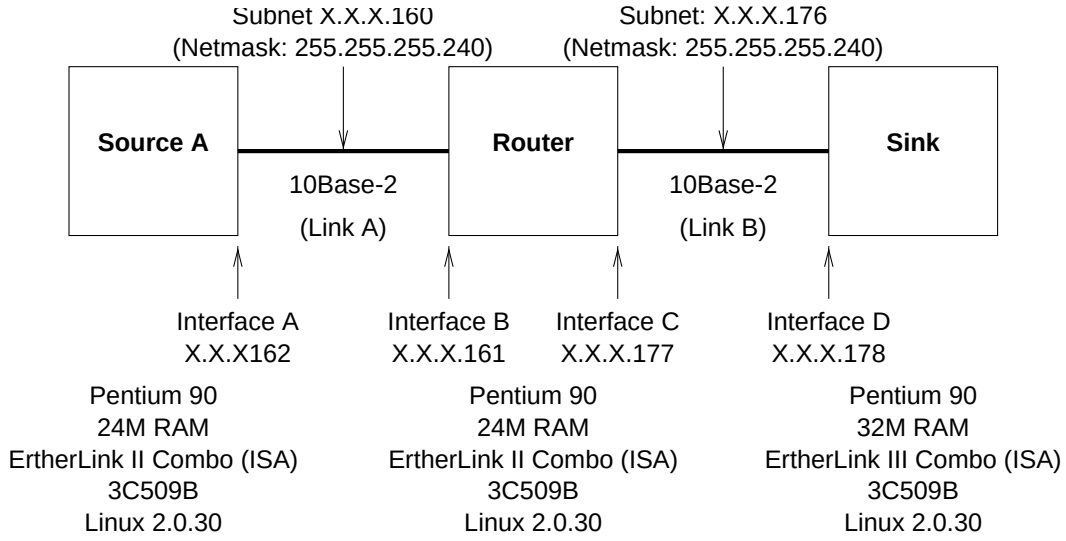


UDP

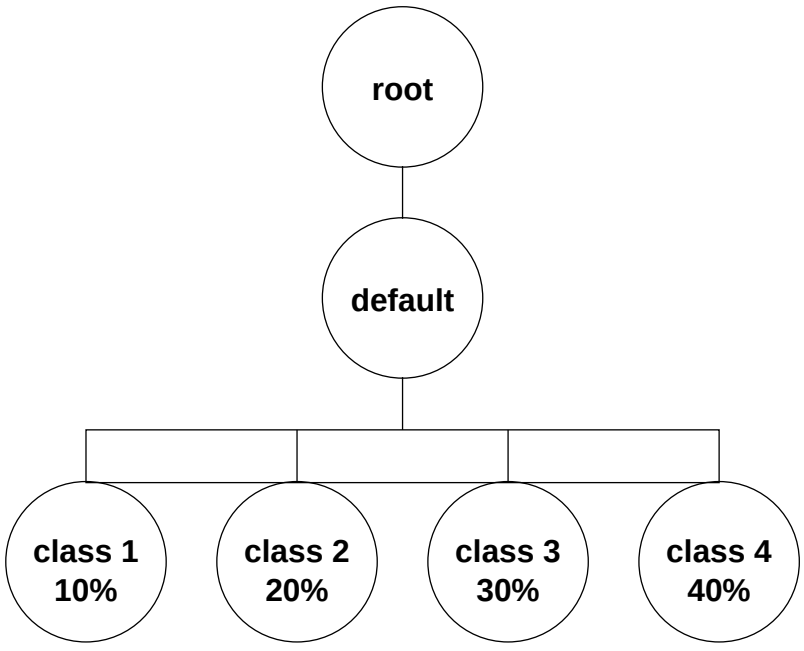


TCP

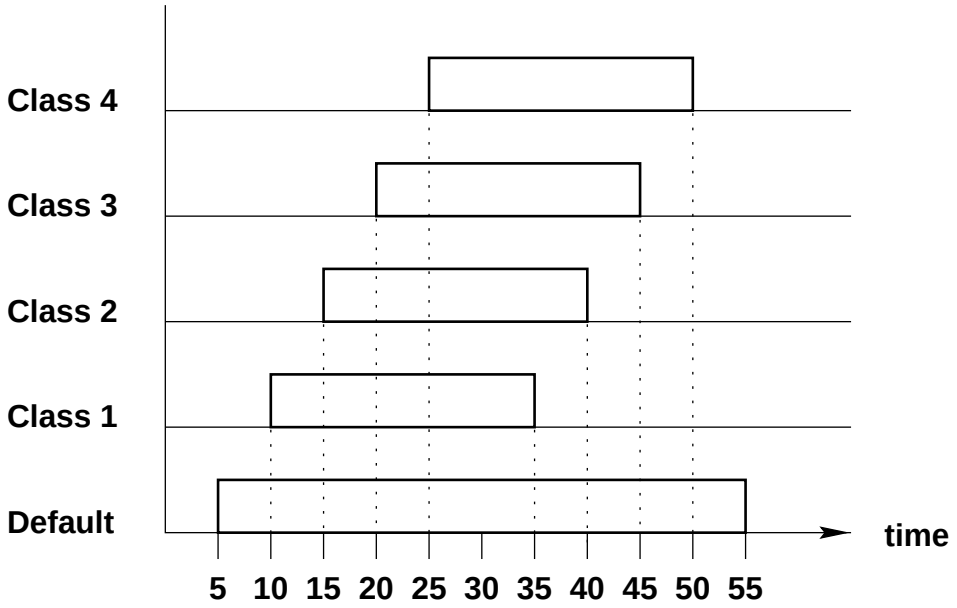
Network Setup for Testing CBQ at End-hosts and Router



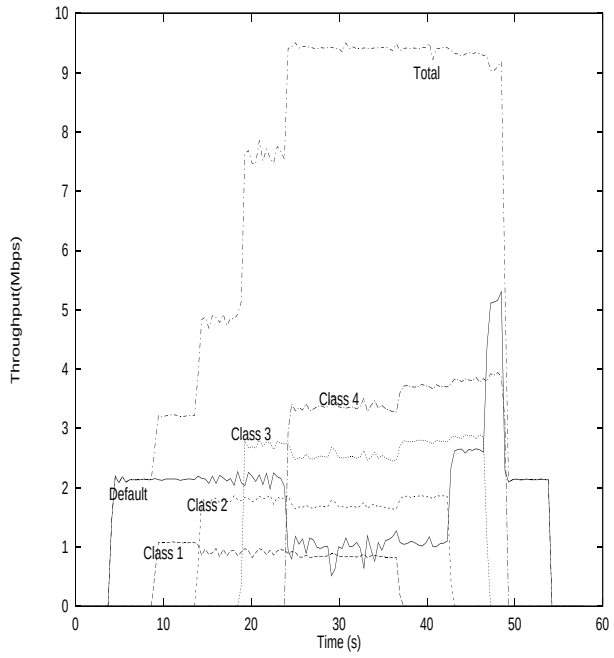
Classes for Multiple UDP Flows



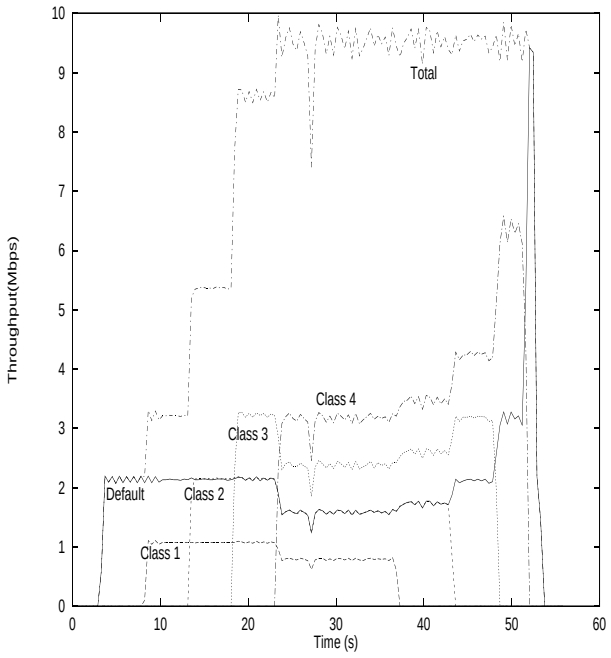
Timing Diagram



Bandwidth Guarantee for UDP Traffic

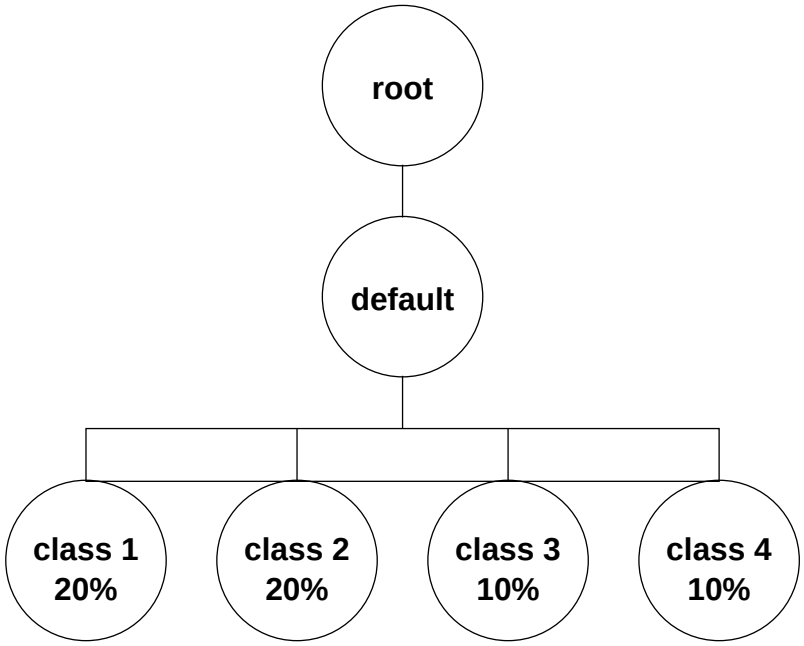


CBQ enabled

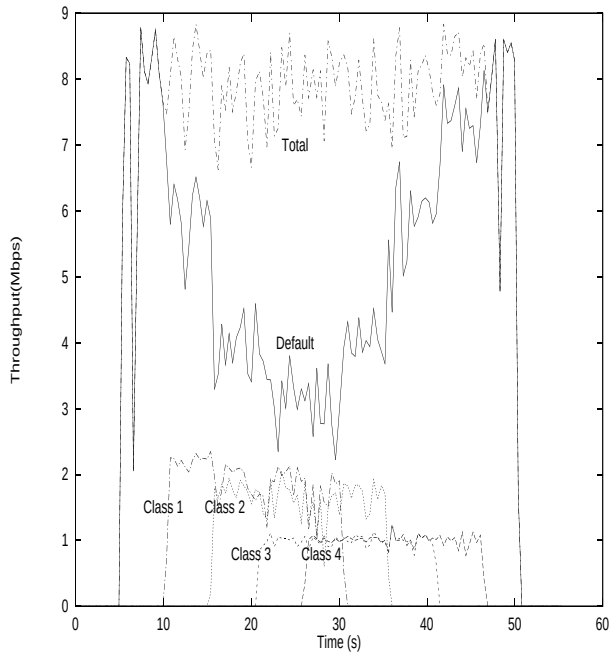


CBQ disabled

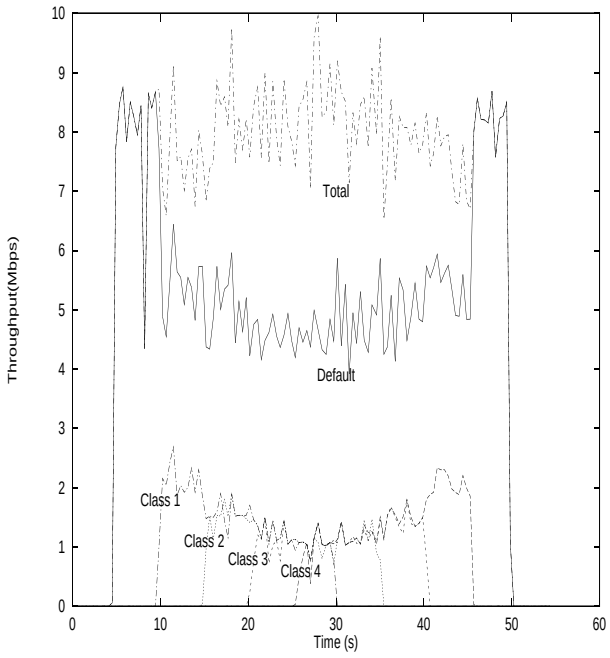
Classes for Multiple TCP Flows



Bandwidth Guarantee for TCP Traffic

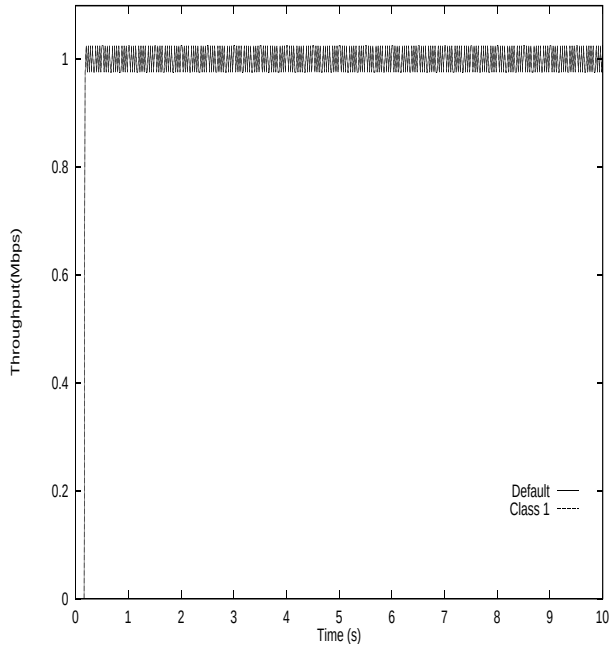


CBQ Enabled

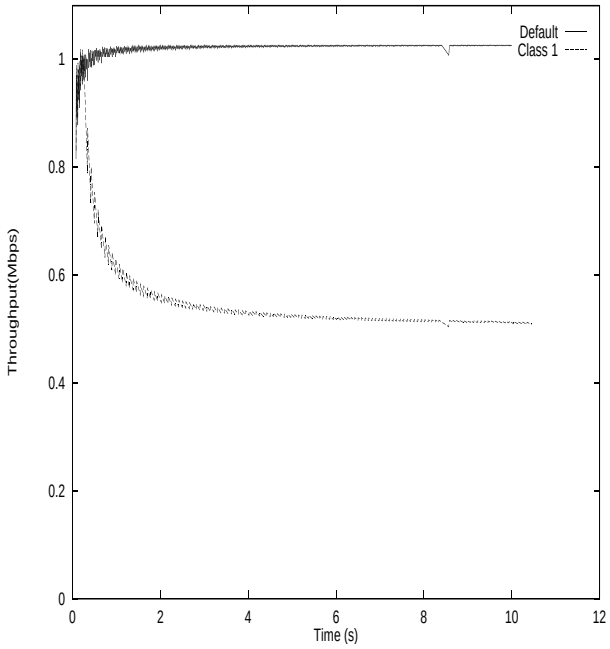


CBQ Disabled

Throughput of UDP Flows

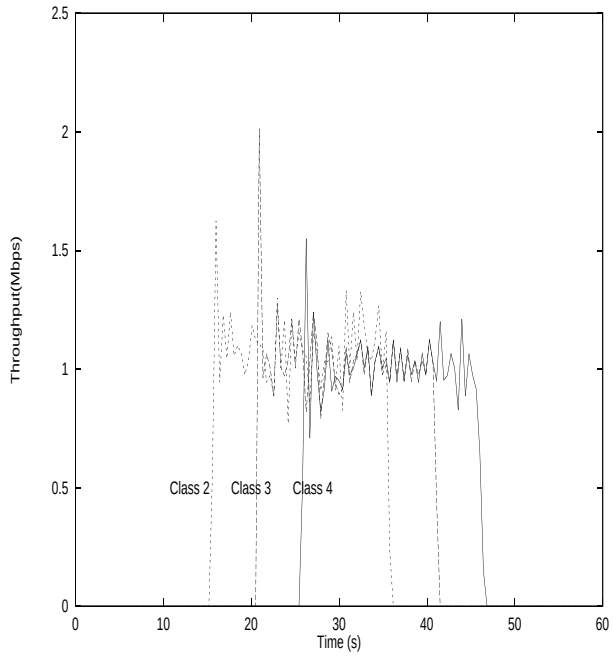


Link A

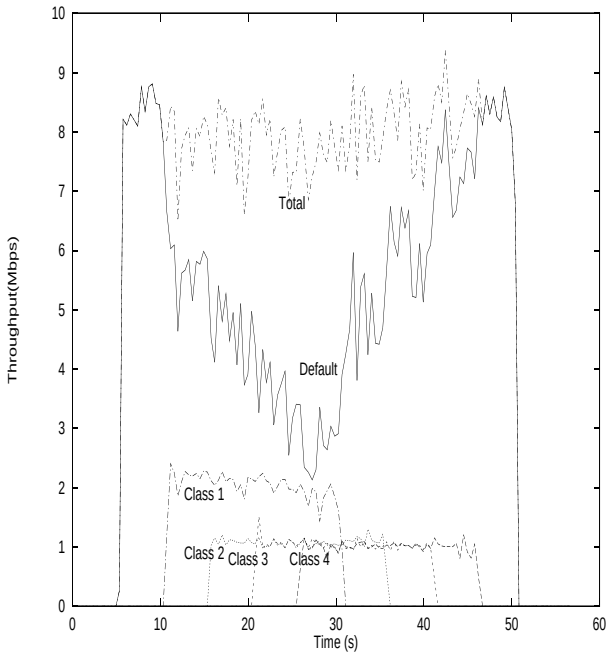


Link B

Throughput of TCP Flows

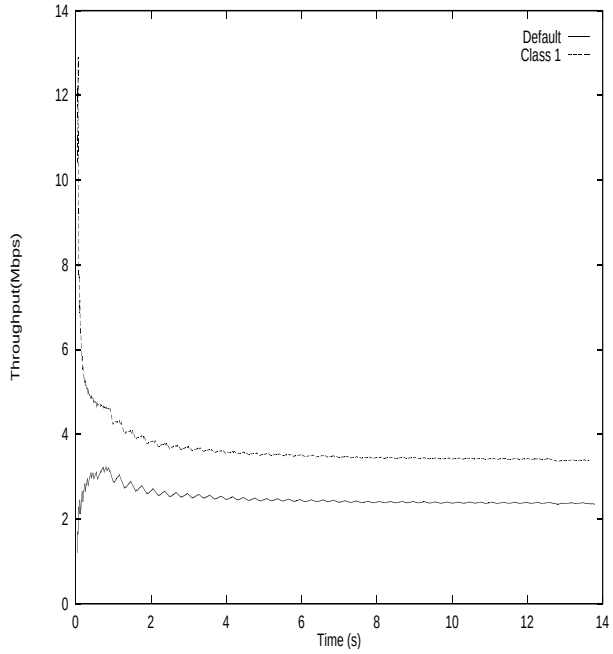


Link A

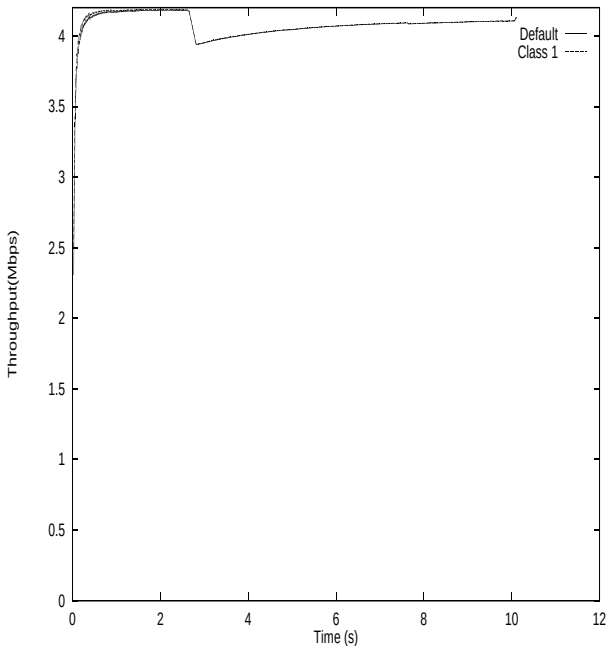


Link B

Throughput of Two UDP Flows

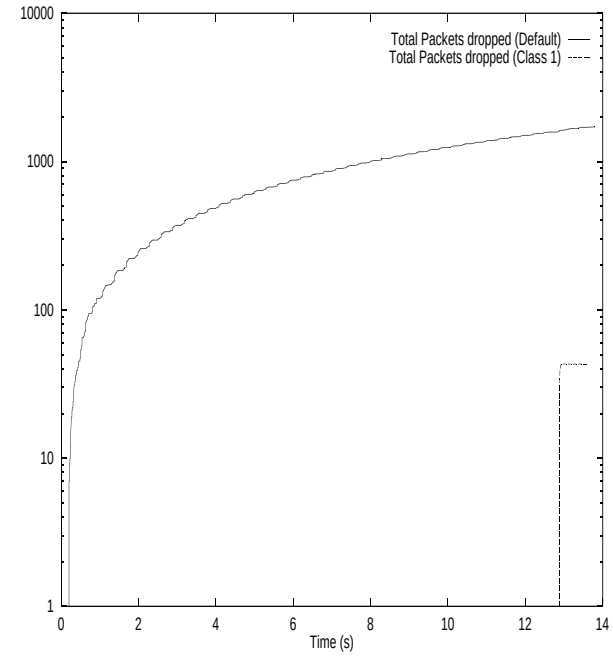


CBQ Enabled

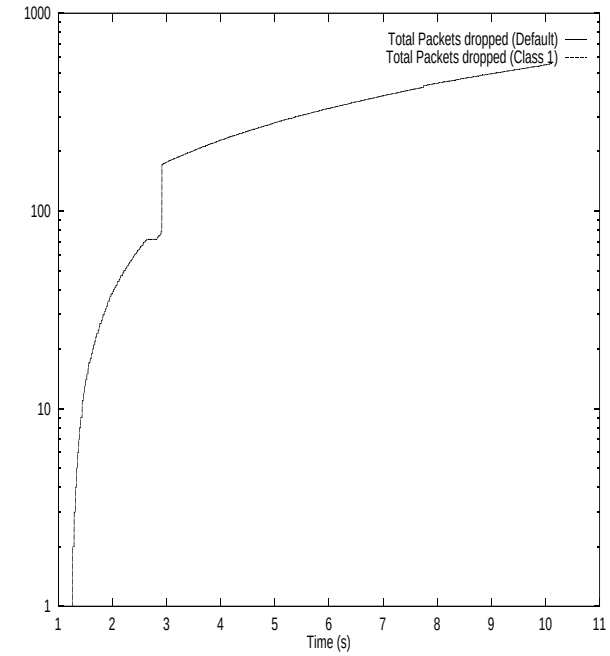


CBQ Disabled

Total Packets Dropped

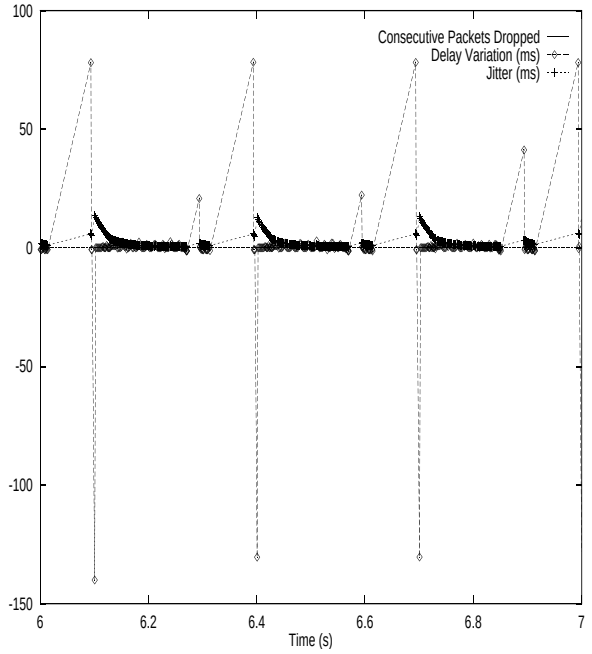


CBQ Enabled

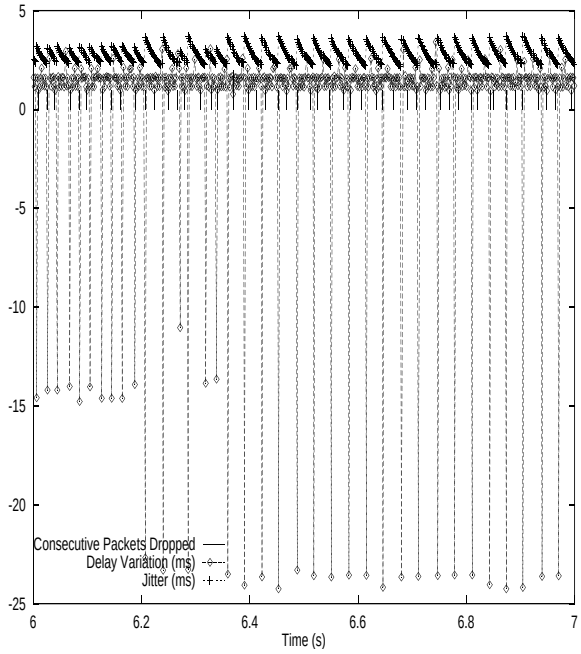


CBQ Disabled

Delay Jitter

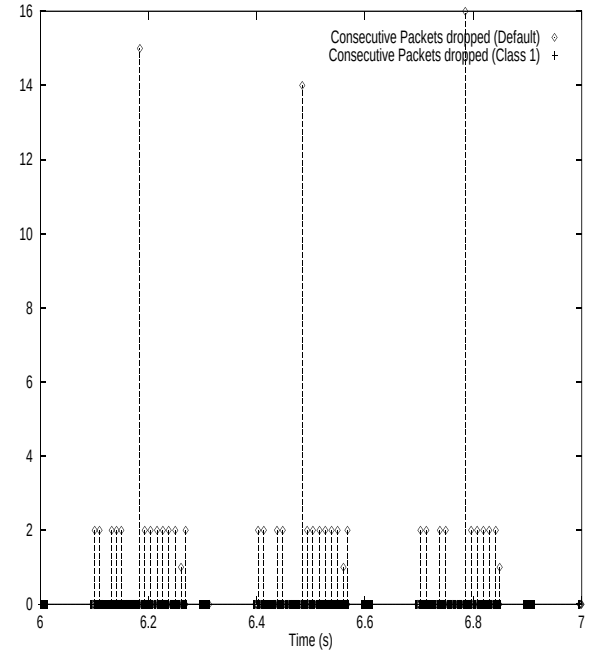


CBQ Enabled

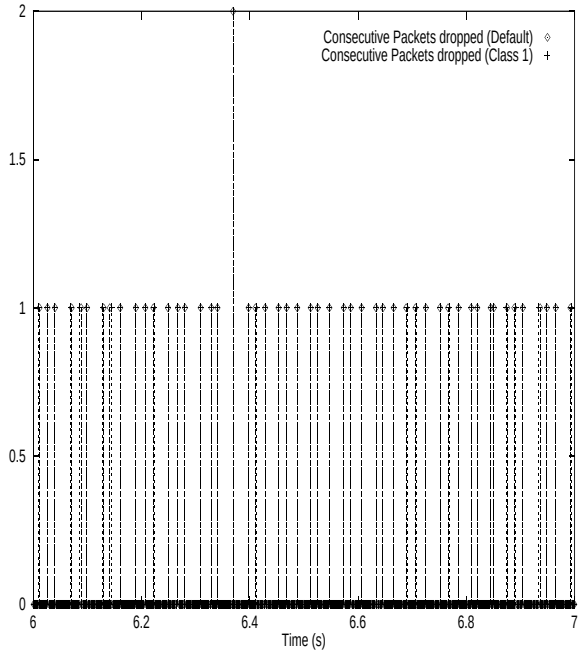


CBQ Disabled

Consecutive Packets Dropped

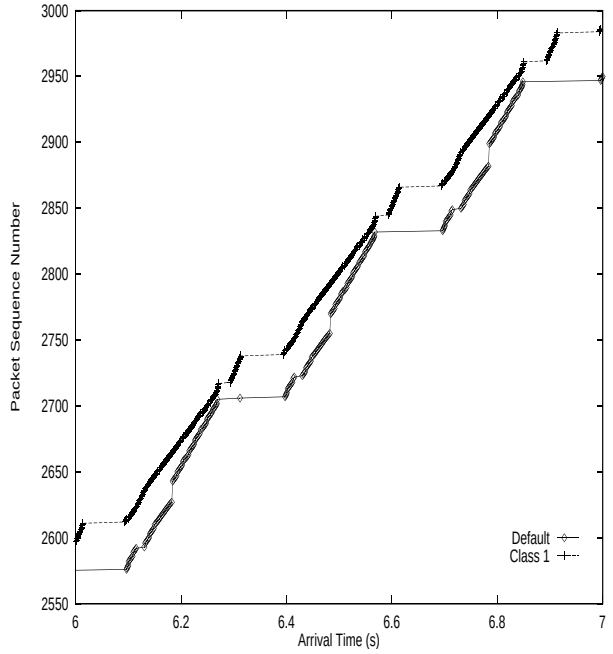


CBQ Enabled

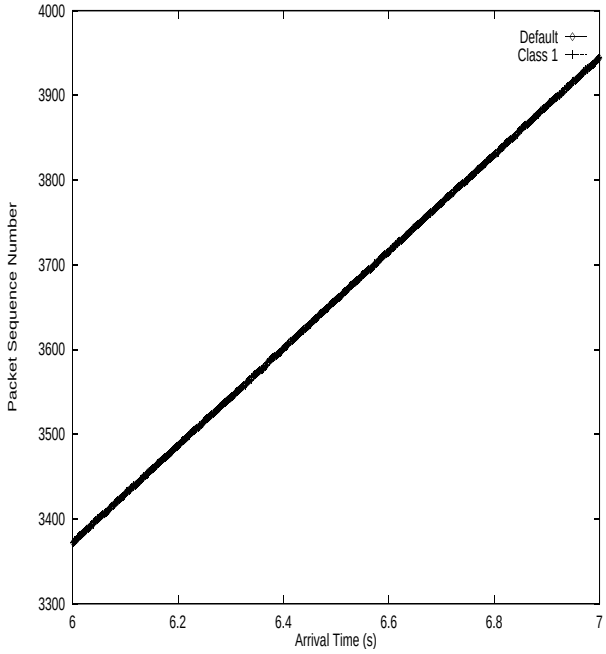


CBQ Disabled

Arrival Times

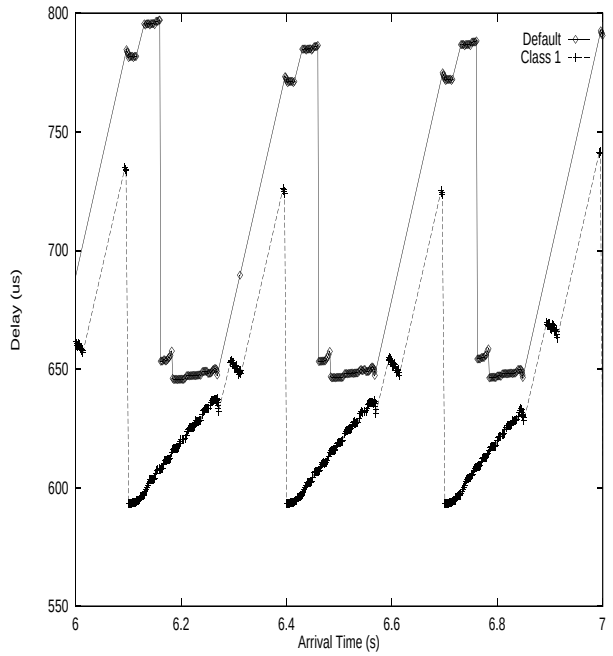


CBQ Enabled

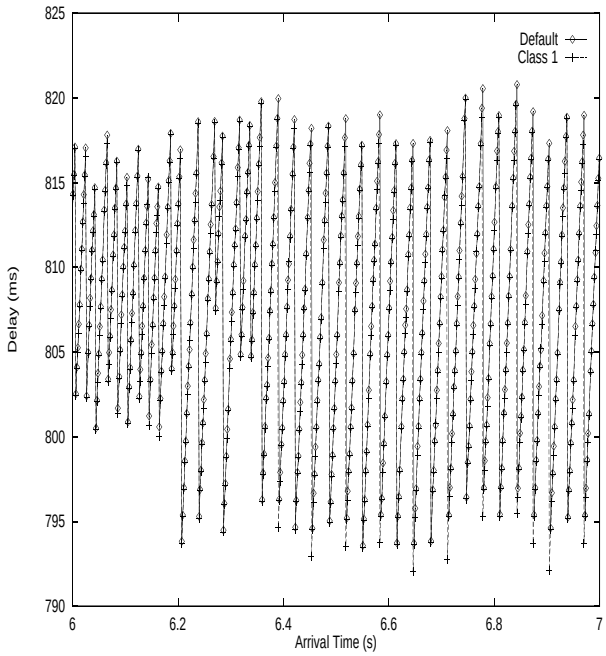


CBQ Disabled

Transmission Delay

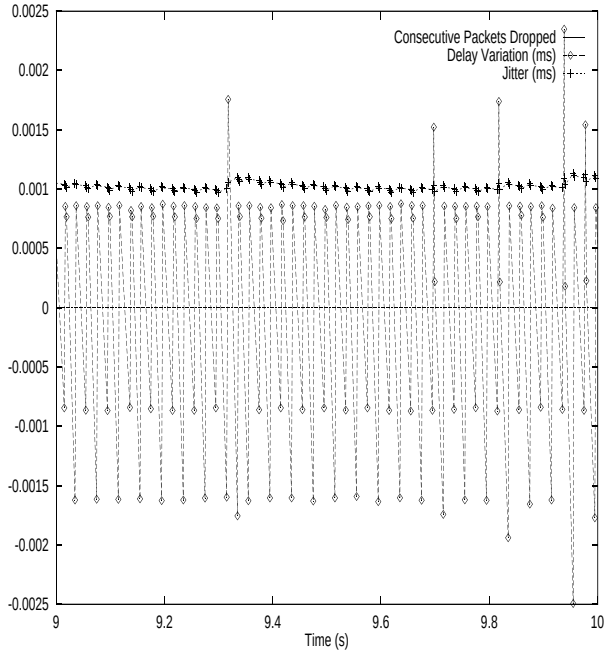


CBQ Enabled

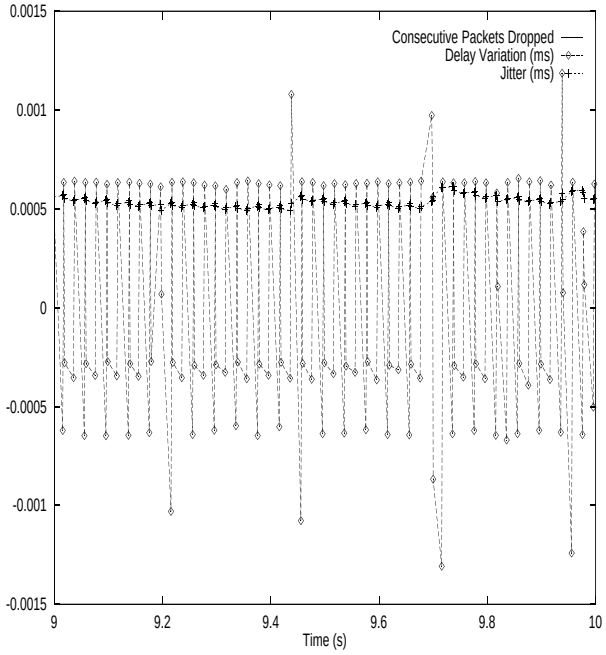


CBQ Disabled

Delay Jitter (with CBQ)

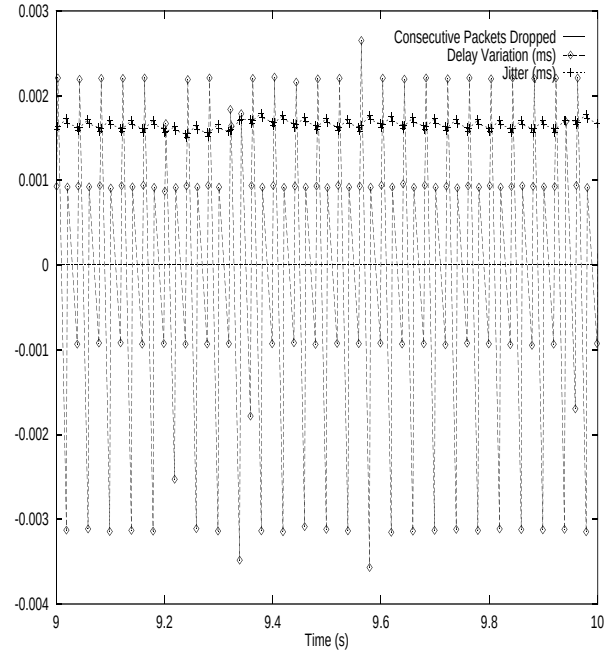


Default Class

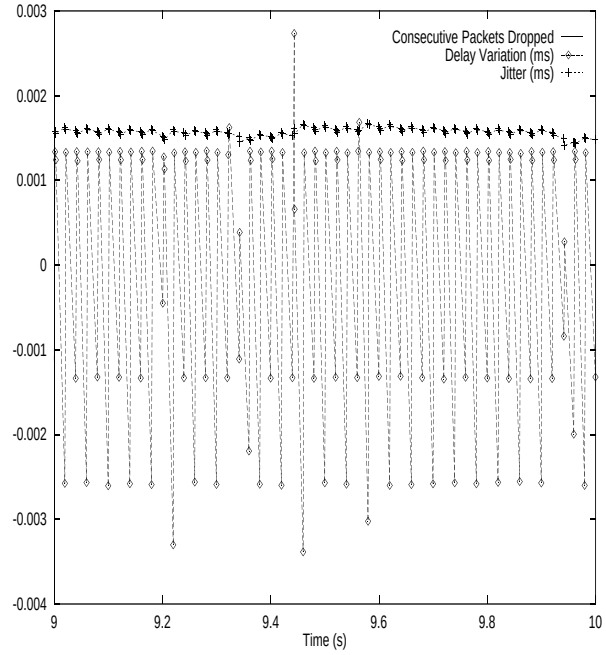


Class 1

Delay Jitter (without CBQ)

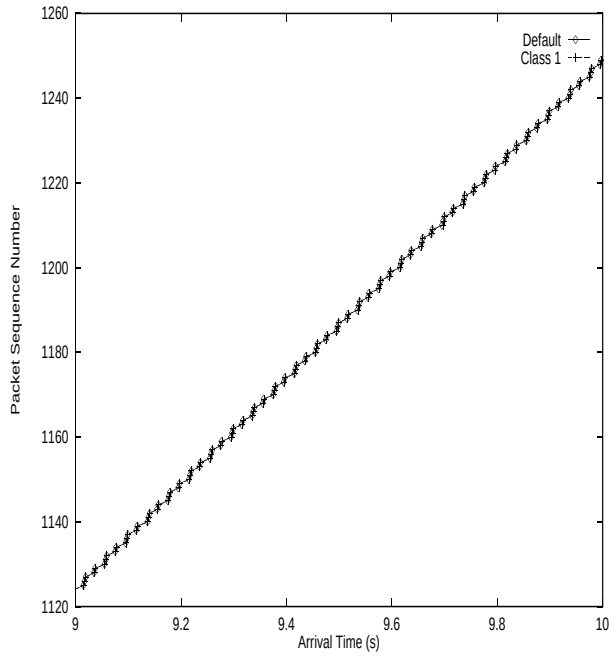


Default Class

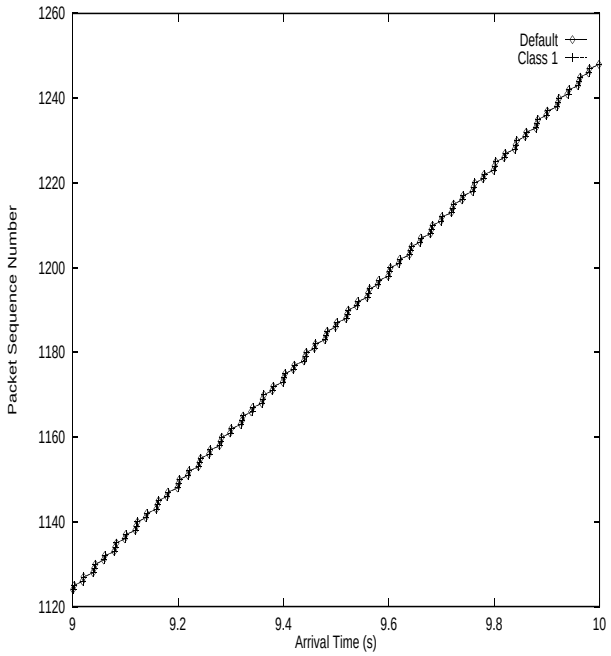


Class 1

Arrival Times



CBQ Enabled



CBQ Disabled

Conclusion

- Able to provide bandwidth guarantee.
- Provides lower delay to higher priority queues.
- Lower packet loss for higher priority queue.
- Required at end-hosts and routers.
- Does not scale with number of flows.
- Could introduces delay jitter if not properly configured.

References

- **S. Floyd**, Pointers to CBQ related papers and implementations. *http://ftp.ee.lbl.gov/floyd/cbq.html*
- **S. Floyd, V. Jacobson**, Link Sharing and Resource Management Models for Packet Networks, *IEEE/ACM Transactions on Networking*, 3(4):365-386, Aug 1995
- **K. Cho**, ALTQ for FreeBSD *http://www.csl.sony.co.jp/person/kjc/programs.html*
- **K. Cho**, A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX based Routers, *Annual Technical Conference, USENIX, 1998*
- **KJ Loh, Irwin Gui, KC Chua**, Performance of a Linux Implementation of Class Based Queueing, *ICCCN '98*

References

- **Werner Almesberger**, Linux Traffic Control – Implementation Overview, *ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz*